Redora: Analysing the mechanisms of stock prediction with Deep Learning and Machine learning

By Daniel Goncalves 29/01/2018 (Stamford International University)

ABSTRACT

Financial firms across the globe currently make use of Deep learning models to more or less accurately predict stock prices, the demand for such software is high and with the age of cryptocurrency there's no doubt the demand shall continue to grow. Gone are the days of mathematicians sitting down and calculating future price trends, in today's world computers can do this for us. Although we know that machines can create good estimates of price predictions, not many of us know what goes on behind these complex algorithms. My research aims to clarify stock prediction from a digital point of view, I shall be looking in depth into Regression, Neural Networks, LSTM networks and how all this comes together into calculating predictions that we can rely on. Although stock prediction with machine learning models has been done many times before we aim to combine multiple data sources and integrated automation so that my software does the trading for you.

1 INTRODUCTION

Within this project scope I shall be focusing on developing and understanding how financial predictions are estimated with cutting edge deep learning and machine learning techniques which are in demand today. I shall carry out a series of tests and demonstrate my results as the research continues. Deep learning is a broader part of machine learning it comprises of a collection of algorithms used to model high level abstractions, with include transformations and complex computations for analysing substantial amounts of data.[1] The deep learning and machine learning topics that we shall look into revolves around Vector Regression, Neural Networks and Recurrent LSTM neural networks.

To validate the successfulness of a prediction made by my software I shall be making use of at least 3 different data sources to train my network on, I shall also use Linear, RBF and polynomial regression models which will also give me a more validated result upon my Recurrent neural network prediction. Due to stock prices constantly changing I plan to make use of Alpha Vantage's API to extract live bitcoin data prior to training my network. Latter given enough time available I shall look into automating my system to make calculated trades with the use of automation scripting in order to imitate a trade that would be made by a real user.

2 RELATED WORK

This kind of research project is not the first of its kind there have been multiple attempts in predicting stocks using the techniques that I shall be using. What makes my project different is that I shall not only be using neural networks for prediction, rather I am combining them to

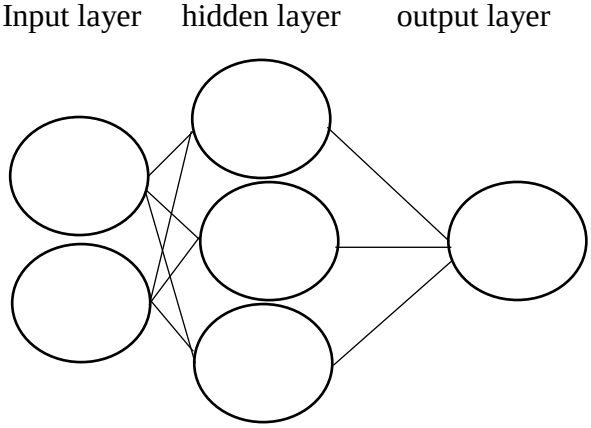
[1] https://www.techopedia.com/definition/30325/deep-learning

make an informed decision, I shall also be using multiple streams of learning data that would reduce the probability of my training data becoming poisoned and producing bad results. My software is also different in the way that it makes real internet trades for the user by formulating estimated forecasts and decisions on whether to buy stocks or sell them. Related work to what I'm trying to do can be seen in commercial products like Ninja Trade and AvaTrade. There are also freelance lecturers like Siraj Raval which coach the Deep Learning Nano Degree at Google, who have demonstrated immense knowledge in the field and has helped me drastically in understanding how RRN's work in predicting stocks.

3 UNDERSTANDING NEURAL NETWORKS AND PREDICTION

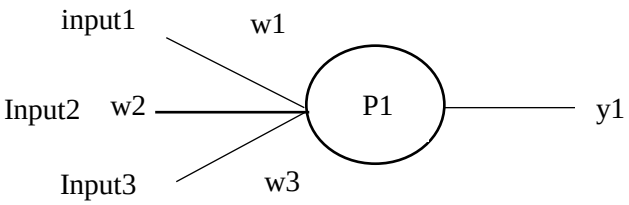
Neural networks can be defined as highly interconnected perceptron's which are based on the model and structure of biological neural networks in our brains.[2]

Perceptron's the most important components which make up these neural networks, this is a type of neuron developed in the 1950 -1960s by Frank Rosenblatt, this was developed over the original idea that Warren McColloch and Walter Pits put forward.[3] A basic Neural Network is made from 3 layers which is the Input Layer, Hidden Layer and the Output Layer. As seen in the diagram each of the lines attached to the perceptron consists of a weight and a bias. These values change drastically during the process of **propagation** for the network to predict a more or less accurate representation of what it believes is the correct value.



Basic neural network, usually we have many hidden layers in most cases

Forward propagation is the process of input training data passing forward through the neural network, through the hidden layer until it gets to the output layer. Forward propagation is straight forward, it constitutes of matrixes of data that need to be squashed using an activation function, typical activation functions include Sigmoid, Tanh, Relu and recently the Leaky Relu! The mathematical transformations can be seen in the diagrams below:



Each of the input axons to the perceptron is multiplied by its weight, then the sum of all weights and inputs is squashed with an activation function and a bias is added. This is done to each perceptron in the hidden layer.

[2] https://www.techopedia.com/definition/5967/artificial-neural-network-ann

[3] http://neuralnetworksanddeeplearning.com/chap1.html

The weighted sum is what we first calculate using the formulae below:

Weighted sum = ((Input 1 * w1) + (Input2 * w2) + (input3 * w3)) can be simplified like,

$$X = \sum_{i=1}^{n} X_i W_i$$

After calculating the weighted sum, we apply an activation function of our choice to squash the weighted sum before we add a bias we do this with the formulae below:

Take g(x) as our activation function:

$$g\left[\sum_{i=1}^{n} X_i W_i\right]$$

The above function normalizes our data by squashing it within the range 0 to 1, this makes normalization of inputs much easier as they pass through the network, the next stage is to add a bias to the above equation, this rids all inequality amongst values from perceptron's it's also part of normalizing data, the biases are first to change in backwards propagation.

$$g\left[\sum_{i=1}^{n} X_i W_i + \text{BIAS}\right]$$

As mentioned before there are a few different Activation functions we can use to squash our inputs and weights, these are Sigmoid, Tanh, Relu and Leaky Relu. You can typically use any function you desire as an activation function; however, we should avoid using an activation function which has a vanishing gradient as this would cause the results to not be accurate since when the network reaches a certain point the gradient disappears leaving all results to be the same. The activation functions mentioned can be seen below:

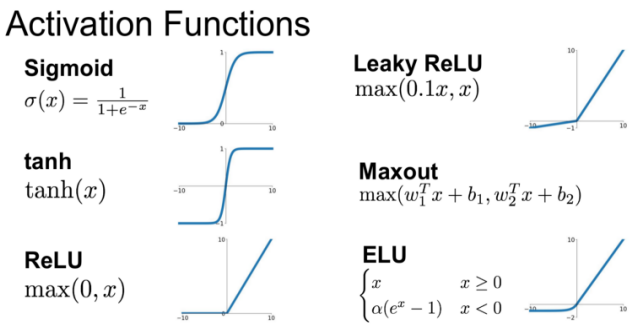
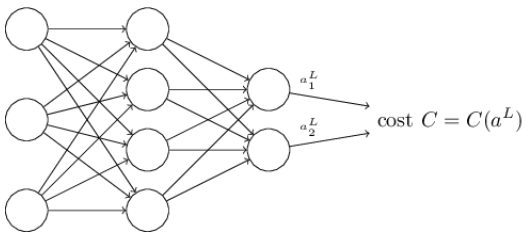


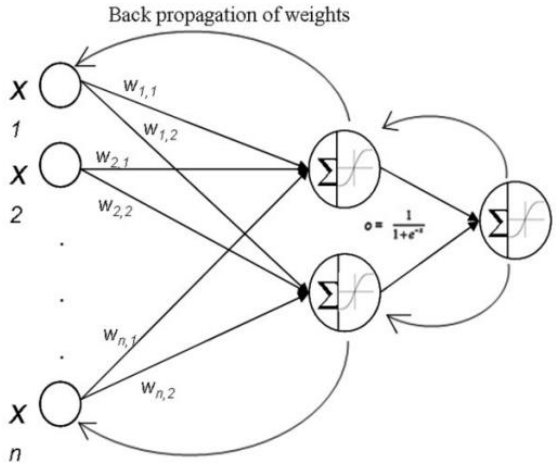
Image derived from [Andrey Nikishaev]

The sigmoid function was the first activation function to be widely adopted in early days, however due to its vanishing gradient it is less widely used now. Tanh also faces similar problem of a vanishing gradient just like the sigmoid function. The ReLU and Leaky ReLU are the simplest yet most effective activation functions in use today as it solves the issue of a vanishing gradient, not only are they effective however they are also easy to make use of.

Back Propagation, this is an algorithm used for adjusting weights and biases in a backwards fashion for supervised learning. This backpropagation mechanism allows the network to correct itself resulting in a better and more accurate result. The more we repeat forward and backwards propagation, the more accurate our result will become. The algorithm makes use of a process called Stochastic Gradient Decent. This will be explained in brief latter.



Above is an image displaying how cost is derived, image derived from Michael Nielsen



Above is an image displaying back propagation, derived from open I.

The process of backpropagation involves reducing cost in the network, cost is determined by utilizing the cost formulae which can be seen below:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

I shall not explain the mathematics behind backpropagation since it involves partial derivation and calculus, I will instead provide formulae's. The below are important formulae's:

The chain rule is very important when doing back propagation. The following is the formulae to get the full derivative of the cost function

∂C/∂w^(l) = 1/n ∑^(n-1) v (k = 0) ∂Ck/∂w^(l)

The formulae to get derivative of our activation:

∂a(l)/∂w^(l) = σ(z^(l)) where z^(l) = w^(l) a^(l-1) + b^(l)

The formulae to get cost of a single neuron path:

$$\textbf{1}\ \frac{\partial Co}{\partial w^{(l)}} = \frac{\partial z^{(l)}}{\partial w^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial CO}{\partial a^{(l)}}$$

$$= a^{(l-1)}\, \sigma(z^{(l)}) 2(a^{(l)} - y)$$

Gradient Decent is used for optimization in our neural network. Gradient decent is the negative gradient of our networks cost function. This optimization can be graphically displayed in the diagram seen below:

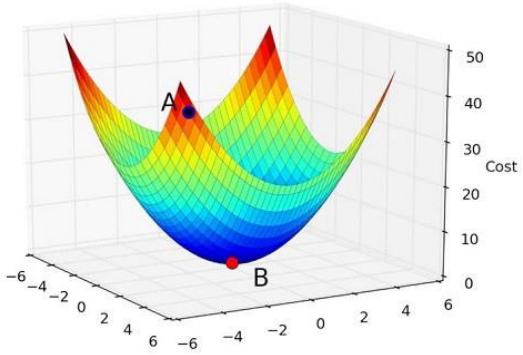


Image derived from Abdullah Al Imran

This is mainly used during the process of Backpropagation where our algorithm needs to adjust weights and biases to predict a more accurate result during the next iteration of sequences. It is the first order iterative for finding the minimum of a cost function. Although we can find the gradient decent of our network by using datasets in their original sequence, this is not a good method of doing so since its slower and takes more time to calculate the gradient decent, instead we make use of randomized datasets which we create mini matches to use in the gradient decent formulae, this method of finding gradient decent is known as **"Stochastic Gradient Decent".** This method greatly improves the learning rate of our network; however, we need to have large datasets to make it work to its full potential.

LSTM networks, these are similar to that of normal neural networks however it consists of LSTM cells, these cells repeat themselves in loops therefore we refer to them as Recurrent Neural Networks. What makes LSTM networks unique in comparison to other networks is that instead of passing the output to the next layer it passes these entire hidden layer, this not only makes way for a better overall result however it is known for its accuracy in forecasting since it has inbuilt memory and is able to remember long term dependencies. The issue of long term dependencies can both be a blessing and a nightmare, because of this sometimes the system may get confused with old data and create an inaccuracy in predictions' cells consist of a variety of gates in which data is processed and passed through, these can involve some complex mathematic computations since a single cell performs all computation. Below is an image displaying both an unrolled LSTM cell and a rolled LSTM cell:

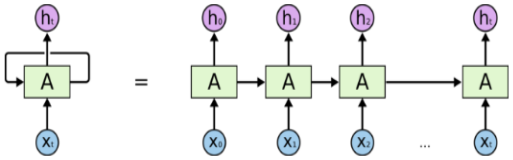


Image derived from Colah , the left side depicts a unrolled LSTM cell and the right side depicts a unrolled LSTM cell.

LSTM networks consist of several logic gates which are seen in the diagram below:

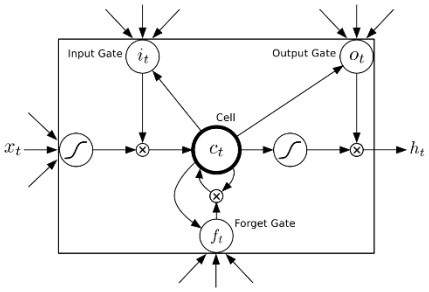
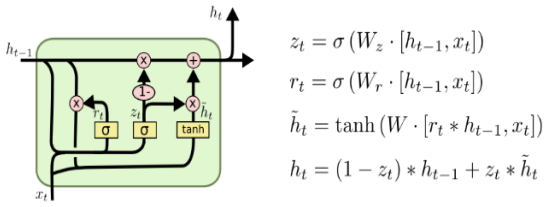


Image from a stack overflow post

There are a number of variants in LSTM network cells, as LSTM networks developed scientists have come up with more workable models one of which is the GRU (gated Recurrent Unit) displayed below:



The following image was derived from Colah , reading this blog will help you understand the dynamics behind he LSTM cell as well as the mathematics explained step by step.

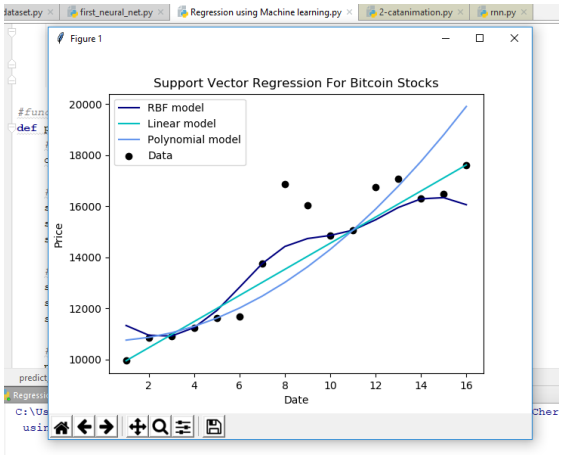
4 ANALYSIS AND ARCHITECHTURE

As part of analysing the mechanics behind stock prediction I have enlisted some milestones which I aimed to accomplish in the given time frame:

- Create a simple SVM regression model to predict data given a past data set
- Create a simple neural network to understand how the data flows through the network and how weights are changed to make predictions better
- Collect data from 3 different data sources in real time using Alpha vantage as one of the data providers
- Build an LSTM network to fully make predictions and graph it using matplotlib
- Make the software make stock purchases based on prediction

Creating a vector regression model

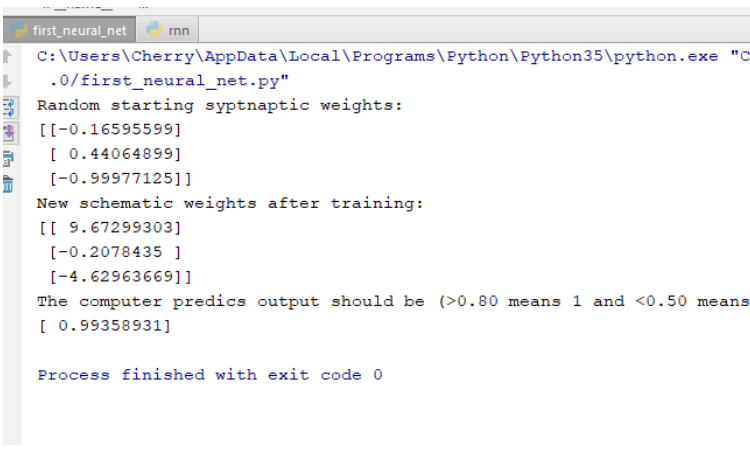
This involved using dependencies like sk.learn, numpy, matplot lib and csv. SK learn is a machine learning library that allows you to create and use preprogramed regression models and datasets for research purposes. SVM regression s basically dividing of data into sections and predicting a line based on the data plots centre. In my project I read past data of bitcoin prices from a csv file and fitted my SVM models (linear, RBF and polynomial) with this data. I then went on to plot this data using matplot lib on a graph to see the direction in which my model is predicting. Numpy was used to format the data from the csv file and separate them into readable columns. The results were as follows:



It is important to note results may vary upon the dataset you use; the above dataset was bitcoin prices extracted from Yahoo Finance between the dates 1 December to 16 December 2017. This was written in python after following some tutorials and guides on creating SVM models all links can be seen below in the reference. The link to the code can be viewed at:

Creating a basic neural network

Due to my limited knowledge in neural networks I decided to build a neural network from scratch without using any machine learning libraries. I used logic gates as my input data, for example AND OR EXOR. I used these because it can easily help us in understanding how the network predicts. For my activation function I made use of a sigmoid, since this activation function does not require extra modification to the output layer, where is if I made use of a RELU or a Leaky Relu I would have to use a SoftMax formulae on the output making the network more complex especially If not using any libraries to assist. I used numpy for manipulation to my figures. It is important to note that this is a very simple neural network only consisting of a single hidden layer, I also use a basic formula for calculating the backpropagation error to adjust the weights. My program consists of the activation function as a method, the activation function derivative as a method and training as a method. The sigmoid method represents our activation function where is the derivative represents derivative of the activation and the training method consists of both the training and backpropagation to adjust weights. Whilst building this neural network it gave me a clear picture on how neural networks work, [Siraj Raval a teacher at Google Nano degree program] helped me a lot through tutorials that he makes for both YouTube and for Google. My results from the neural network were as follows:

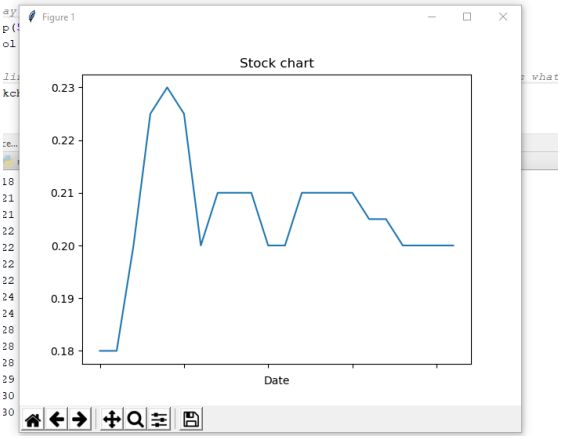


The test was successful, what the program basically did was feed the network with 1s and 0s for logic gates and then based on this data is predicted the next value based on past data, the network predicted a 99 % chance that the output value should be a 1 and it was indeed supposed to be a 1, any value lower than 0.50 would mean that the network thinks the value is a zero. This experiment helped me in understanding the way weights are adjusted and how the network works.

Collecting data from 3 different APIs and data sources

I decided to collect data from alpha vantage API since this enabled me to stream live bitcoin data to my program, I also made use of Bit stamp which I accessed through their API. And finally, I got the third data source from a csv file filled with data collected from Google Finance. In order to sort the data extracted using the APIs I used JSON in which I pulled only the data that I desired to display in my program. I also used dependencies like Matplot lib to plot my data and sleep to enable breaks in between the data collections. The JSON library was used to decrypt the JSON from the APIs mentioned. The graph was plotted after each iteration from collecting the data from the 3 sources, the results can be seen below:

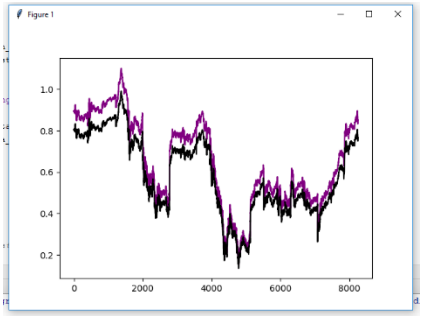
Date	close	high	volume	open	low
2018-01-16 18:00:00	0.180	0.180	11275.0	0.180	0.180
2018-01-16 19:46:00	0.180	0.180	22973.0	0.180	0.180
2018-01-16 22:46:00	0.200	0.200	7000.0	0.200	0.200
2018-01-16 22:47:00	0.225	0.225	3000.0	0.225	0.225
2018-01-17 18:00:00	0.230	0.230	7782.0	0.230	0.230
2018-01-17 19:08:00	0.225	0.225	1565.0	0.225	0.225
2018-01-17 20:37:00	0.200	0.200	4000.0	0.200	0.200
2018-01-18 19:36:00	0.210	0.215	7694.0	0.215	0.210
2018-01-21 23:01:00	0.210	0.210	952.0	0.210	0.210
2018-01-21 23:28:00	0.210	0.210	28374.0	0.210	0.210
2018-01-22 18:34:00	0.200	0.200	4000.0	0.200	0.200
2018-01-22 20:22:00	0.200	0.200	1000.0	0.200	0.200
2018-01-22 22:12:00	0.210	0.210	21626.0	0.210	0.210
2018-01-22 22:52:00	0.210	0.210	18057.0	0.210	0.210



The results proved to be moderately successful, considering data was being loaded in real time, however I did face the challenge of normalizing the data since all three APIs where giving out different stock prices it was hard to believe which one was correct and which was wrong, I think this is a major issue as bad data can poison your network into predicting wrong prices.

Creating a LSTM network to predict stock prices based on past data

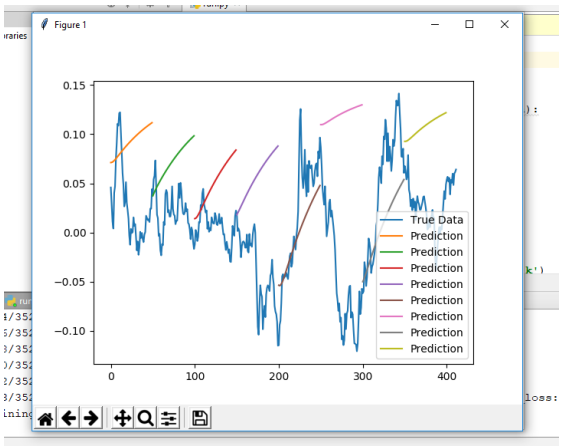
This was a major challenge however with the help of various research papers and YouTube channels I managed to create the network. Firstly I decided to try use TensorFlow to build a neural network and plot both actual data and predictions, this was successful after following Sebastian Heinz tutorial, I built it however I modified his existing code not to display the minibatches instead I only wanted to display the predicted prices along with actual prices to suit the use case of my program, the results from the Tensor flow were as follows:



The black line represents actual price data and the purple line represents the networks predicted price. In this program I made use of dependencies like TensorFlow, Numpy, pandas, sklearn.preprocessing, matplotlib. The results of this model displayed acceptable level of accuracy as we can see the difference between actual and expected is not that great.

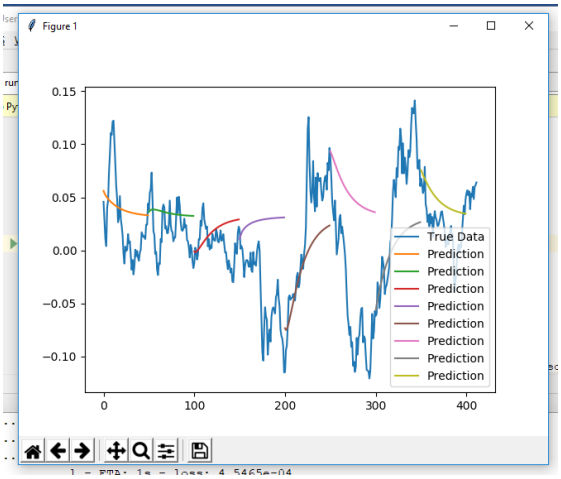
I also created a LSTM network using TensorFlow, in which I conducted various experiments to improve training of the model. My experiments included changing the values for Epoch and sequence values. The model that I worked on was based on a timeseries, the model predicts curves as the time series goes on. My results for different settings where as seen below:

Settings for LSTM model 1:
epoch:1
sequence length: 50



The settings for this model predicted mainly upwards curvature, although the model seems to work it has some trouble predicting downwards patterns. I decided to increase the epoch and maintain the sequence length. By increasing the epoch, we increase the number of training iterations, therefore prior to increasing the epoch I was in expectance that the results would be better. Results for my second model can be seen below:

Settings for LSTM model 2:
epoch:10
sequence length: 50

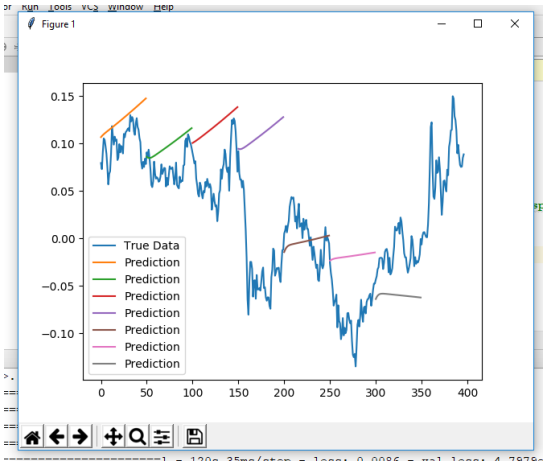


As we can see above my expectations where right by increasing the number of training iterations (epoch) we achieve much better results. I kept the sequence length at 50 just to normalize my results for comparison purposes between model 1 and model 2. These predictions where a good sign as now we can see both upwards curvature and downwards curvature. In my next model I decided to keep epoch at 1 and increase the training time, I decided to do this just to see how much impact the sequence length has on the prediction of a time-series. My results can be seen in the model below.

Settings for LSTM model 3:

epoch:1

sequence length: 200

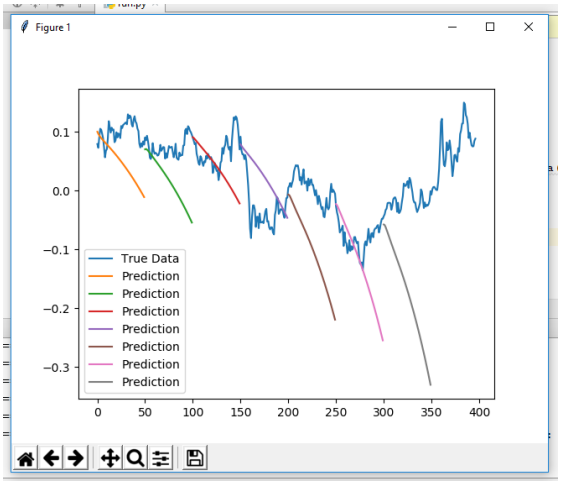


As we can see above there is not much curvature in prediction, and results look separated upwards trend on one side and downwards on the other. The predictions where also less in this model and there was a lot of loss experienced in the network. The delay for prediction was also long, therefore defeating the purpose of timely predictions. I believe there should be a balance between the epoch and the sequence length, however we should not set the sequence length too high as this will delay our results drastically, therefore decreasing performance of our network. In my next experiment I decided to increase the epoch in proportion to the sequence length just like what we did in model 2 except with a sequence length of 200. I did this too see if setting a same proportion except with higher values has any impact on the predictions themselves. The results can be seen below.

Settings for LSTM model 4:

epoch:40

sequence length: 200



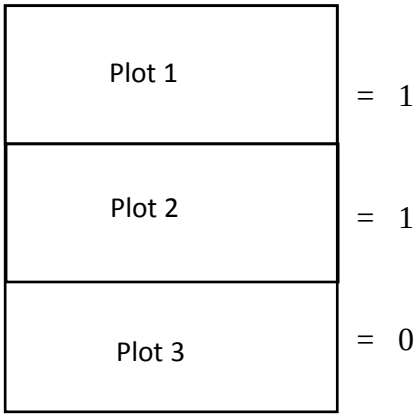
The results for this test as we can see where not good, all predictions are facing downwards. Therefore, from the results we can justify that the epoch when put into proportion with the sequence length does not have any effect on the prediction outcome. I was expecting the results to come out similar to that of my LSTM model 2, however this was not the case. Also, do not attempt to train your model with these settings if you have a large dataset, after training my model with over 3000 data entries and with the settings above, my computer took 1 hour 5 minutes to compute the predictions and almost lead my computer to crashing as it was using 100% of resources available.

Conclusion from LSTM network analysis

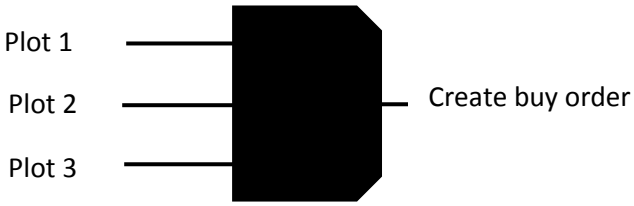
After analysing all model tests, I conducted I can conclude that I shall be using model 2 of my 4 models to predict the time series of predictions. The reason being the results of prediction are the closest out of 4 tests conducted, in addition the time in which it takes to predict stocks is relatively ok in comparison to other models, it does not stress the computer too much and can run on a machine comprising of 4Gig RAM and icore 5 processor and above.

Automation of stock predictions

I have decided to implement 3 graph plots to my final program all on a single canvas. Each of the graphs hold an equal say in predicting the stocks. I decided to feed each of my models with a different data source, this will allow for more reliable data, instead of predicting stocks based on a single source of data I shall be using three different sources plotted on three different graphs. The way in which I have implemented it is such that my system basically works like an AND GATE, since we have 3 graphs if any 2 of the graphs predicts that a stock price will rise it will automate a stock order through the IQ option website. Automation will be done by using a library called win32 api, this will allow us to initiate a click event anywhere on the screen. I have decided to use this library since the website we want to make predictions on is very secure and does not allow you to view button id's and classes. By using this automation library, we just input an x axis and a y axis on the screen in which to initiate a click event and it shall complete it. Below is a model of how my prediction mechanism works based on the three graphs:



My model shall follow a similar scenario to the AND gate in terms of Logic. For a buy purchase to be made, at least 2 of any of the charts need to be predicting that the price will rise before the buy order to automated. This allows for a more accurate prediction especially since each of the data sources are different, this allows for data to be more informed and it also concours the problem of where one of your data sources may be poisoned with invalid data and a wrong prediction will be made. It solves the problem since 2 of the three datasets need to have bad data to affect the trade order.



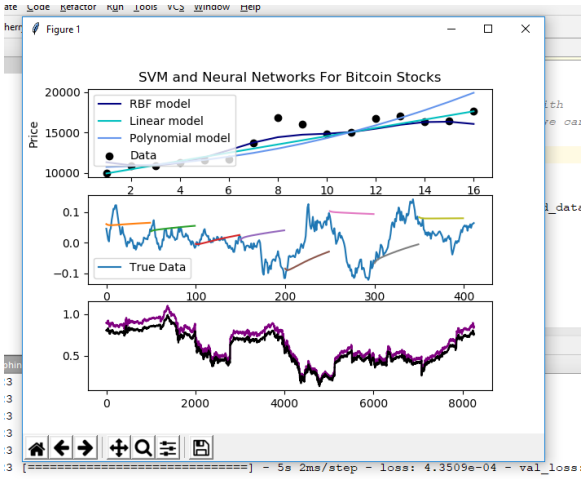
Plot 1	Plot 2	Plot 3	Order
1	0	0	0
0	1	0	0
0	0	1	0
1	1	0	1
1	0	1	1
1	1	1	1
0	1	1	1

5 IMPLEMENTATION

After I successfully achieved all milestones, in order to execute my idea of having a single program predict and buy stocks automatically I combined all three tables into a single window, in which the program analyses results and decides on whether to predict or not. In my opinion I think solved the problem that I intended to, which was the trade of stocks automatically based on a variety neural networks and datasets.

My program managed to do just that. Although the prediction seems to be good, I did face some issues which I will

mention soon, the image below is a screenshot of my program:



As we can see from the screenshot of my final program above, each graph is slightly different from each other, this is because 3 different datasets were used. This was to get a more accurate result as outcome, although this helped a lot, it also caused some issues.

6 ISSUES FACED

During the development of my program I did face various issues which I have enlisted below:

- The data sources I used varied significantly leading me to not know which data source is correct.
- Since the predictions are based on downloaded csv data sources, there is a problem of real time prediction. Even if we use real time data the time taken for a prediction to occur is long and the prediction would therefore be wrong by the time its predicted.
- The program is very heavy on resources and cannot work efficiently on computers with few resources.
- During testing results take along time to be predicted leading me to wasting time.

7 RELAVENCE

I believe my program shall serve some relevance for other researchers and financial analysts that may want to further develop my solution. There is a lot of room for improvement, however I believe I paved a way for predictions to take place. The area of my research is a very lucrative market which can be taken advantage of providing models like this are developed further. Times are changing, and financial stock centres are now moving towards analytic computational predictions and are therefore counting on the data analysts to produce models that can efficiently predict stocks.

8 FUTURE WORK

Given time in the future I shall be working on improving the architecture of my model sufficiently to cater for most worst-case scenarios which may occur, I shall be focusing on the following in particular:

- Reducing time delays by implementing my solution on the cloud instead of local hosting it
- Developing a more concrete model which can dynamically change itself based on a variety of data inputs
- Developing an attractive GUI to provide cutting edge “eye candy” for the users of my system.
- I will look into using more real time sources of data with limitation to a past time period, since neural networks need be trained current data may not work well for the situation.

- Thank you for reading up, if you have any interests for collaboration or other research projects that I may be able to help with, please feel free to reach out to : danielgoncalves62@gmail.com

------THE END-----

References

https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html

http://neuralnetworksanddeeplearning.com/

https://medium.com/@xenonstack/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7

http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

http://neuralnetworksanddeeplearning.com/chap1.html

https://deeplearning4j.org/neuralnet-overview

https://medium.com/machine-learning-for-humans/neural-networks-deep-learning-cdad8aeae49b

https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877

https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html

http://tradingtuitions.com/python-script-plot-live-stock-chart/

https://medium.com/machine-learning-world/neural-networks-for-algorithmic-trading-1-2-correct-time-series-forecasting-backtesting-9776bfd9e589
https://medium.com/machine-learning-world/neural-networks-for-algorithmic-trading-part-one-simple-time-series-forecasting-f992daa1045a

https://medium.com/abdullah-al-imran/intuition-of-gradient-descent-for-machine-learning-49e1b6b89c8b

https://www.youtube.com/watch?v=p69khggr1Jo

http://neuralnetworksanddeeplearning.com/chap3.html